Pipes

- Built-in Pipes
- Using parameters
- Chaining Pipes
- Custom Pipes
- Pipes and Change Detection
- Pure and Impure pipes

Built-in Pipes

A pipe takes in data as input and transforms it to a desired output.

Angular comes with a stock of pipes such as

1. **DatePipe**: Formats a date according to locale rules
2. **UpperCasePipe**: Transforms string to uppercase
3. **LowerCasePipe**: Transforms string to lowercase
4. **DecimalPiple**: Formats a number according to locale rules.
5. **CurrencyPipe**: Formats a number as currency using locale rules.
6. **PercentPipe**: Formats a number as percentage.
7. **JsonPipe**: Converts value into string using JSON.stringify. Useful for debugging.
8. **SlicePipe:** Creates a new List or String containing a subset (slice) of the elements

Parameterizing a Pipe:

A pipe may accept any number of optional parameters to fine-tune its output. We add parameters to a pipe by following the pipe name with a colon (:) and then the parameter value (e.g., currency:'EUR'). If our pipe accepts multiple parameters, we separate the values with colons (e.g. slice:1:5)

DatePipe: date_expression | **date**[:format]

format values:

- 'medium': equivalent to 'yMMMdjms' (e.g. Sep 3, 2010, 12:05:08 PM for en-US)
- 'short': equivalent to 'yMdjm' (e.g. 9/3/2010, 12:05 PM for en-US)
- 'fullDate': equivalent to 'yMMMMEEEEd' (e.g. Friday, September 3, 2010 for en-US)
- 'longDate': equivalent to 'yMMMMd' (e.g. September 3, 2010 for en-US)
- 'mediumDate': equivalent to 'yMMMd' (e.g. Sep 3, 2010 for en-US)
- 'shortDate': equivalent to 'yMd' (e.g. 9/3/2010 for en-US)
- 'mediumTime': equivalent to 'jms' (e.g. 12:05:08 PM for en-US)

- 'shortTime': equivalent to 'jm' (e.g. 12:05 PM for en-US)

DecimalPipe: number_expression | **number**[:digitInfo]

- digitInfo is a string which has a following format:
  {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}
    - minIntegerDigits is the minimum number of integer digits to use. Defaults to 1.
    - minFractionDigits is the minimum number of digits after fraction. Defaults to 0.
    - maxFractionDigits is the maximum number of digits after fraction. Defaults to 3.

Eg: number:'3.5-5'

CurrencyPipe: number_expression | **currency**[:currencyCode[:symbolDisplay[:digitInfo]]]

- currencyCode is the ISO 4217 currency code, such as USD for the US dollar and EUR for the euro.
- symbolDisplay is a boolean indicating whether to use the currency symbol or code.
    - true: use symbol (e.g. $).
    - false(default): use code (e.g. USD).
- digitInfo is a string as explained in Decimal Filter

Eg: {{emp.salary | currency:'USD':true:'4.2-3'}}

PercentagePipe: number_expression | **percent**[:digitInfo]

Eg: {{10.12345 | percent:'4.3-5'}}

SlicePipe: array_or_string_expression | slice:start[:end]

- start: The starting index of the subset to return.
    - **a positive integer**: return the item at start index and all items after in the list or string expression.
    - **a negative integer**: return the item at start index from the end and all items after in the list or string expression.
    - **if positive and greater than the size of the expression**: return an empty list or string.
    - **if negative and greater than the size of the expression**: return entire list or string.
- end: The ending index of the subset to return.
    - **omitted**: return all items until the end.
    - **if positive**: return all items before end index of the list or string.
    - **if negative**: return all items before end index from the end of the list or string.

NOTE: We can chain pipes together in potentially useful combinations.

Employee.ts

```
export class Employee
{
   id: number;
   name: string;
   salary: number;
   dateOfJoin: Date;
   constructor(id: number, name: string, sal: number, doj: Date)
   {
      this.id = id;
      this.name = name;
      this.salary = sal;
      this.dateOfJoin = doj;
   }
}
```

Employees.component.ts

```
import { Component } from '@angular/core';

import { Employee } from './employee';

@Component({
   moduleId: module.id,
   selector: 'employees',
   template: `
      <style>
      table, th, td {
         border: 1px solid black;
      }
      </style>
      <table>
         <tr *ngFor="let emp of employees | slice:3:6">
            <td>{{emp.id}}</td>
            <td>{{emp.name | uppercase}}</td>
```

```
            <td>{{emp.salary | currency:'USD':true:'4.2-3'}}</td>
            <td>{{emp.dateOfJoin | date:'yMMMdjms'}}</td>
            <td>{{10.12345 | percent:'4.3-5'}}</td>
          <tr>
      </table>
    <hr/>
    <pre>{{employees | json }} </pre>
`
})
export class EmployeesComponent {
    employees = [
      new Employee(1, "Raj Kumar", 11000.12345, new Date(1980, 5, 13)),
      new Employee(2, "Sudheer Reddy", 12000, new Date(1982, 15, 23)),
      new Employee(3, "Kiran Rao", 13000, new Date(1983, 25, 32)),
      new Employee(4, "Kishor Kumar", 14000, new Date(1983, 25, 3)),
      new Employee(5, "Usha Rani", 12000, new Date(1985, 25, 31)),
      new Employee(6, "Sai Kiran", 12300, new Date(1986, 26, 36)),
      new Employee(7, "Jacob John", 11000, new Date(1987, 27, 33)),
    ]
}
```

Custom Pipes

- A pipe is a class decorated with pipe metadata.
- The pipe class implements the PipeTransform interface's transform method that accepts an input value followed by optional parameters and returns the transformed value.
- There will be one additional argument to the transform method for each parameter passed to the pipe.

1. Add to project a new class AgePipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';


@Pipe({ name: 'age' })
export class AgePipe implements PipeTransform {
    transform(date: Date, format: string): string {
      let todayOffset = new Date().getTime();
      let pardateTimeOffset = date.getTime();
```

```
    let diff = todayOffset - pardateTimeOffset;
    let oneDay = 1000 * 60 * 60 * 24
    if (format == null || format.toLowerCase() == "y")
      return Math.floor(diff / (oneDay * 365)) + " Years";
    else if (format.toLowerCase() == "m")
      return Math.floor(diff / (oneDay * 31)) + " Months";
    else if (format.toLowerCase() == "d")
      return Math.floor(diff / (oneDay)) + " Days"
  }
}
```

Custom pipes must be registered manually. We must include our pipe in the declarations array of the AppModule.

2. Edit app.module.ts

```
import { AgePipe } from './age.pipe';
and
declarations: [AppComponent, EmployeesComponent, AgePipe ],
```

To use the Pipe

3. Edit employees.component.ts

```
import { AgePipe } from './age.pipe';
and
<td>{{emp.dateOfJoin | age:'m'}}</td>
```

Parameterized Custom Pipe

Eg 2: Pipe to filter employees having salary greater than specified value:

```
@Pipe({ name: 'salary' })
export class SalaryPipe implements PipeTransform
{
  transform(allEmployees: Employee[], salaryGreaterThan: number): Employee[]
  {
    return allEmployees.filter(emp => emp.salary > salaryGreaterThan);
  }
}
```

Example3: Reversing a String

Now we need to import **PipeTransform** from "@angular/core"

From the above example change the following code in the corresponding file.

File: **app.CustomPipes.ts**

```
import { Pipe,PipeTransform } from "@angular/core";

@Pipe({
   name: "reverse"
})

export class ReverseStringPipe implements PipeTransform {
   transform(value: any, start:any, end:any) {
      var text: string = value;
      var sliceStr: string; var subStr: string;
      if (start != null && end != null) {
         subStr = value.split("").slice(start, end).join("");
         sliceStr = value.split("").slice(start, end).reverse().join("");
         return text.replace(subStr, sliceStr);
      }
      else
      return text.split("").reverse().join("");
   }
}
```

File: **app.component.ts**

Now in the template provide the parameters by separating with colon.

```
template: `
            <h2>Reverse string</h2>
            <p> {{text}} = {{text | reverse:0:5}} </p>
            `
```

Output: Now the output will be as follows

Hello World = olleH World

Now try changing the template parameters

```
template: `
        <h2>Reverse string</h2>
        <p> {{text}} = {{text | reverse:6:11}}</p>
        `
```

Output:

Hello World = Hello dlroW

Pure and Impure Pipes

Pipes are categorized into two types

1. Pure Pipes (**immutable/ not changeable**): A pure pipe will execute only when the input value or the input parameters are changed. Shows better performance.
2. Impure Pipes (**mutable / changeable**): To run a pipe independent of the condition which is applied for pure pipes. We need to set the **pure property to false.** Because impure pipes tracks all the changes, performance is poor when compare to pure pipes because of continuous tracking on the changes.

All the above discussed pipes are **pure pipes**

Example: Creating a new pipe **"join"** and set the **pure property true**.

File: **join.pipe.ts**

```
import { Pipe, PipeTransform } from "@angular/core"

@Pipe({
    name: "join",
    pure: false
})

export class JoinPipe implements PipeTransform {
    transform(value: any) {
        return value.join(",");
    }
}
```

File: **app.component.ts**

```
import { Component } from '@angular/core';

@Component({
    selector: 'my-app',
    template:`
      <input type="text" #txtname />
      <button (click) ="addItem(txtname.value)"> Add Name </button>
      <p> Join array items - {{arr | join }}</p>
    `
})
```

```
export class AppComponent {
    arr = ["Ramesh", "Suresh"];
    addItem(item: any) {
        this.arr.push(item); //here we are only changing the data and not the Array.
    }
}
```

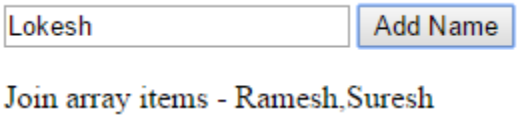
File: app.module.ts

```
import { JoinPipe } from './join.pipe';

declarations: [AppComponent, JoinPipe],
```

Output:

Here if I want to add another item to be added into the array and apply the join pipe it will not show effect.



To make it impure

Let's just change the **pure property to false** and add one more name.

```
@Pipe({
  name: "join",
  pure: false
})
```

Output:

Now if I try to add a new item that will get added and join pipe will work for the new array